## Лабораторная работа № 5.1 (MathCad)

**Темы:**

1. Изучение аффинных преобразований в пространстве.
2. Изучение принципов построения 3D – изображений.
3. Изучение методов удаления невидимых граней для выпуклого многогранника.

**Задание 1 (*Выполняется в пакете MathCad*).**

I. Задать координаты вершин усеченной пирамиды (рис.1) в мировой декартовой системе координат $(X\,Y\,Z)$ и положение камеры (наблюдателя) в мировой сферической системе координат $(r, \varphi, \theta) = (10, 315^\circ, 45^\circ)$.

II. Построить изображение пирамиды без удаления невидимых граней, рис. 1а (использовать лекционный пример). Построенный рисунок должен ***моделировать*** изображение пирамиды в прямоугольной области окна Windows $D^w(x_L^w, y_L^w, x_H^w, y_H^w)$ с координатами:

- $(x_L^w, y_L^w) = (200, 100)$ – оконные координаты левого верхнего угла области $D^w$;
- $(x_H^w, y_H^w) = (700, 500)$ – оконные координаты правого нижнего угла области $D^w$;

III. Построить изображение пирамиды с удалением невидимых граней, рис. 1б (использовать лекционный пример). Построенный рисунок должен ***моделировать*** изображение пирамиды в прямоугольной области окна Windows $D^w(x_L^w, y_L^w, x_H^w, y_H^w)$ с координатами:

- $(x_L^w, y_L^w) = (200, 100)$ – оконные координаты левого верхнего угла области $D^w$;
- $(x_H^w, y_H^w) = (700, 500)$ – оконные координаты правого нижнего угла области $D^w$;

**Задание 2**

IV. Задать координаты вершин усеченной пирамиды (рис.1) в мировой декартовой системе координат $(X\,Y\,Z)$ и начальное положение камеры (наблюдателя) в мировой сферической системе координат $(r, \varphi, \theta) = (10, 315^\circ, 45^\circ)$.

V. Создать приложение Windows для изображения:

- усеченной пирамиды без удаления невидимых граней, рис. 1а;
- усеченной пирамиды с удалением невидимых граней, рис. 2б.

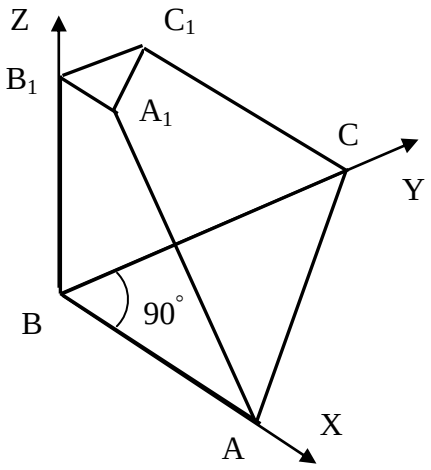



Рис. 1а

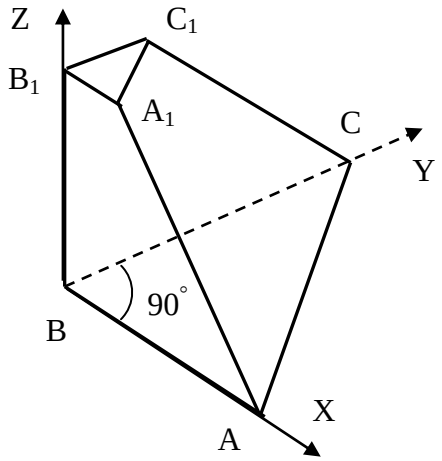



Рис. 1б

В режиме изображения пирамиды с удалением невидимых граней противоположные грани (верхнюю и нижнюю) закрасить разными цветами.

Использовать аксонометрическая проекцию фигуры на картинную плоскость.

Изображения строятся в режиме отображения MM_TEXT.

Координаты вершин задать в конструкторе по умолчанию. Положение камеры (наблюдателя) задаётся в мировой сферической системе координат $(r, \varphi, \theta)$. Начальные значения $(r, \varphi, \theta)$ определяются в конструкторе по умолчанию.

Каждое из изображений пирамиды появляется на экране при выборе соответствующего пункта меню.

Обеспечить изображение фигуры при перемещении камеры по углу $\varphi$ (клавиши «→» и «←») и углу $\theta$ (клавиши «↑» и «↓»).

Обеспечить масштабирование фигуры при изменении размеров окна.

Класс **CPyramid** создается в отдельном файле LibPyramid

```
class
{
    CMatrix Vertices;            // Координаты вершин
    void GetRect(CMatrix& Vert, CRectD&  RectView);
```

```
    // Вычисляет координаты прямоугольника, охватывающего проекцию
    // пирамиды на плоскость XY в ВИДОВОЙ системе координат
    // Vert - координаты вершин (в столбцах)
    // RectView - проекция - охватывающий прямоугольник
public:
CPyramid(); // Конструктор по умолчанию
void Draw(CDC& dc, CMatrix& P, CRect& RW);
    // Рисует пирамиду С УДАЛЕНИЕМ невидимых ребер
    // Самостоятельный пересчет координат из мировых в оконные (MM_TEXT)
    // dc - ссылка на класс CDC MFC
    // P - координаты точки наблюдения в мировой сферической системе //координат
    // (r, fi(град.), q(град.))
    // RW - область в окне для отображения
void Draw1(CDC& dc, CMatrix& P, CRect& RW);
    // Рисует пирамиду БЕЗ удаления невидимых ребер
    // Самостоятельный пересчет координат из мировых в оконные (MM_TEXT)
    // dc - ссылка на класс CDC MFC
    // P - координаты точки наблюдения в мировой сферической системе
     // координат
    // (r,fi(град.), q(град.))
    // RW - область в окне для отображения
};
```

Требуемые функции:

**CMatrix CreateViewCoord(double r,double fi,double q)**
// Создает матрицу пересчета точки из мировой системы координат в видовую
// (r,fi,q)- координата ТОЧКИ НАБЛЮДЕНИЯ (начало видовой системы координат)
// в мировой сферической системе координат ( углы fi и q в градусах)

**CMatrix VectorMult(CMatrix& V1,CMatrix& V2)**
// Вычисляет векторное произведение векторов V1 (3x1) и V2(3x1)

**double ScalarMult(CMatrix& V1,CMatrix& V2)**
// Вычисляет скалярное произведение векторов V1(3x1) и V2(3x1)

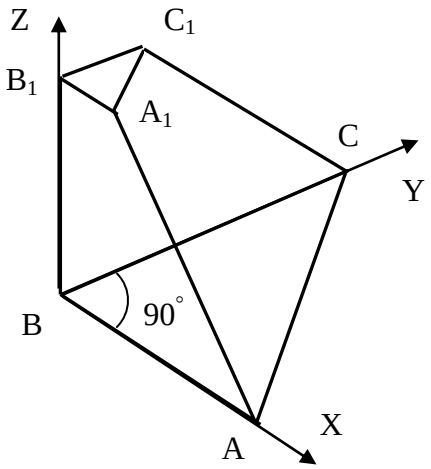



Рис. 1а

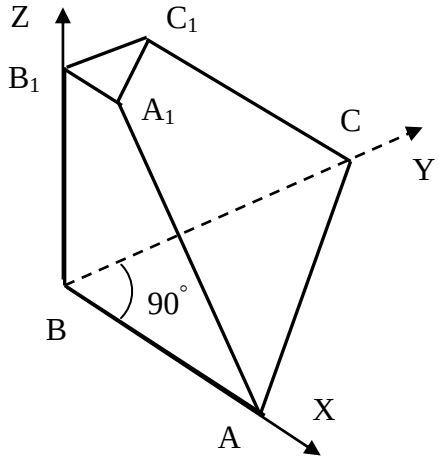



Рис. 1б

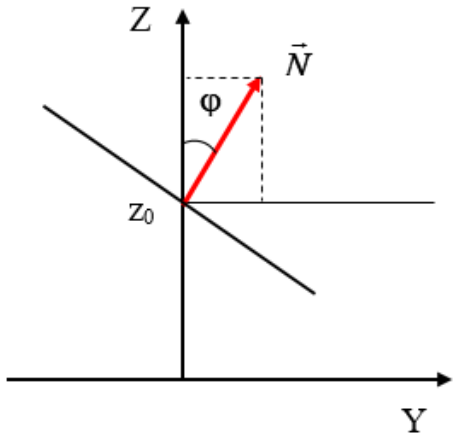
IV. Построить сечение пирамиды (без удаления невидимых граней) плоскостью, положение которой в пространстве определяется:

вектором нормали

- $\vec{N} = \vec{N}(N_x, N_y, N_z) = \vec{N}(0, \sin\varphi, \cos\varphi)$;

точкой пересечения с осью $Z$

- $A_0 = A_0(x_0, y_0, z_0) = A_0(0, 0, z_0)$ .



V. Отобразить изображение в области окна Windows размером:
   - координата левого верхнего угла области отображения
   - (x, y) = (100, 200);
   - координата правого нижнего угла области отображения
   - (x, y) = (800, 900);

**Теоретические сведения:**

**Сферическая система координат.**

Сферическая система координат – трехмерная система координат, в которой каждая точка пространства определяется тремя числами (r, θ, φ), где r — расстояние до начала координат (радиальное расстояние), а θ и φ — зенитный и азимутальный углы соответственно рис. 1.

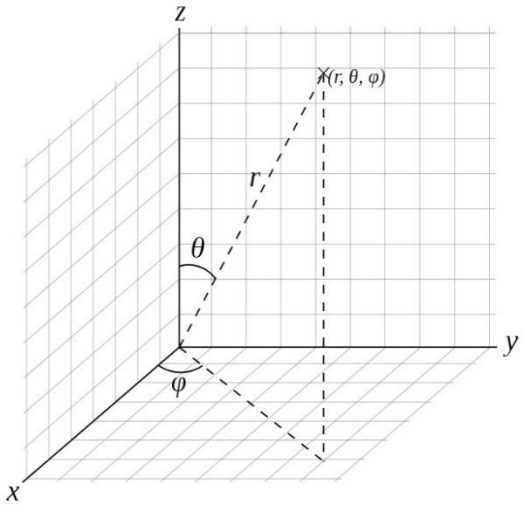


*Рисунок 1- Сферическая система координат*

**Зенит** — направление вертикального подъёма над произвольно выбранной точкой (точкой наблюдения), принадлежащей фундаментальной плоскости.

**Азимут** — угол между произвольно выбранным лучом фундаментальной плоскости с началом в точке наблюдения и другим лучом этой плоскости, имеющим общее начало с первым.

Положение точки P в сферической системе координат определяется тройкой , где:

- $r \geq 0$ — расстояние от начала координат до заданной точки
- $0° \leq \theta \leq 180°$ — угол между осью **z** и отрезком, соединяющим начало координат и точку P.
- $0° \leq \varphi \leq 360°$

**Однородные координаты**

Однородные координаты – это математический механизм, связанный с определением положения точек в пространстве. Привычный аппарат декартовых координат, не подходит для решения некоторых важных задач в силу следующих соображений:

- в декартовых невозможно описать бесконечно удаленную точку.
- не позволяет произвести проверку различия между точкой и вектором.
- невозможно использовать унифицированный механизм работы с матрицами для выражения преобразований точек.
- декартовы координаты не позволяют использовать запись для создания перспективного преобразования точку.

Однородные координаты в двумерном изображении имеют вид (x, y, w), где w- масштабный множитель.

Двумерные декартовы координаты точки получаются из однородных делением на множитель w:

X=x/w, Y=y/w

Основные свойства однородных координат:

- наборы чисел однородных координат могут соответствовать одной точке в Декартовых координатах, например точки: (6, 8, 4) и (3, 4, 2)
- в силу произвольного значения **w** не существует единственного представления точки, заданной в декартовых координатах
- как минимум одно число из трех, не должно равняться 0
- деление на **w** всех координат даст Декартовы координаты (x/w, y/w,1)
- при w=0, точка находится в бесконечности

Все матрицы преобразований имеют размер 3x3. Таким образом матрицы преобразования имеют один и тоже же вид.

**Матрица переноса:**

p’ = p T(Dx,Dy), где:

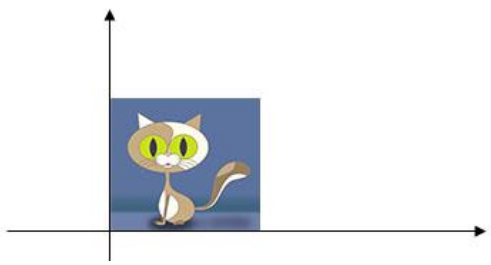


*Рисунок 2- Изображение в начале координат*

p’ – новые координаты

p – старые координаты

$$T(D_x, D_y) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ D_x & D_y & 1 \end{bmatrix}$$

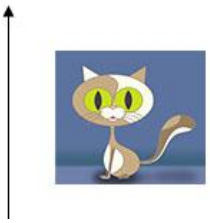


*Рисунок 3- Результат переноса*

$$[x',y',1] = [x, y, 1]\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ D_x & D_y & 1 \end{bmatrix}$$

**Матрица растяжение (сжатия):**

x’ = αx,α>0,

y’ = βy,β>0.

$$Taf = \begin{bmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

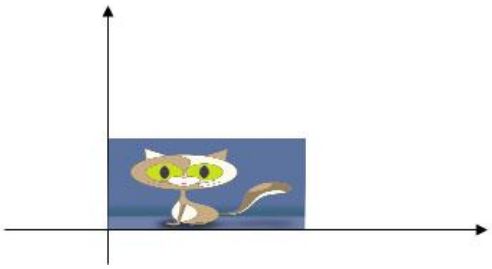
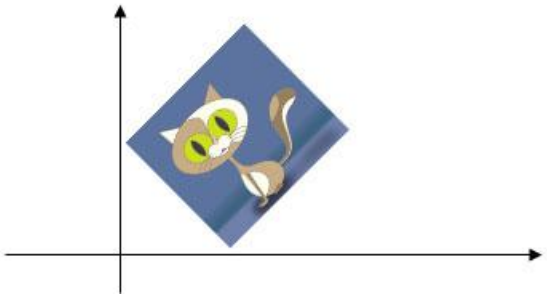


*Рисунок 4- результат работы растяжения/сжатия*

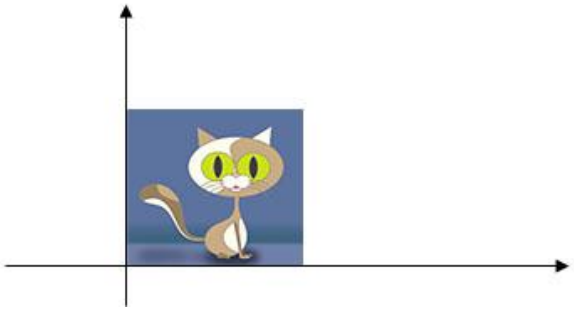
**Матрица вращения относительно центра:**

x’ = xcosφ - ysinφ,

y’ = xsinφ – ycosφ,



$$Taf = \begin{bmatrix} \cos\varphi & -\sin\varphi & 0 \\ \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

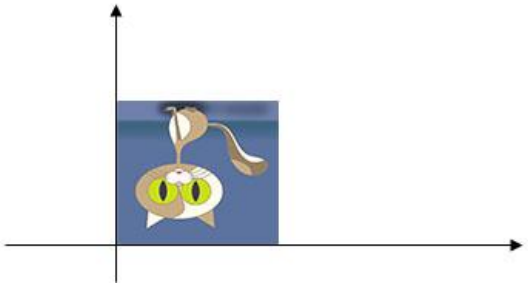


**Матрица отражения:**

**x’ = x (x’ = -x),**

**y’ = -y(y’ = y),**

*Рисунок 6- отражение по оси OY*



$$Taf = \begin{bmatrix} 1 & 0 & 0 \\ \sin\varphi & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$ - ось OY,

$$Taf = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$ – по оси OX

*Рисунок 7-отражение по оси OX*

**Видовая система координат**

Для изображения объекта на экране нужно пересчитать его мировые координаты в другую системы координат, которая связана с точкой наблюдения. Эта система координат называется видовой системой координат и является левосторонней.

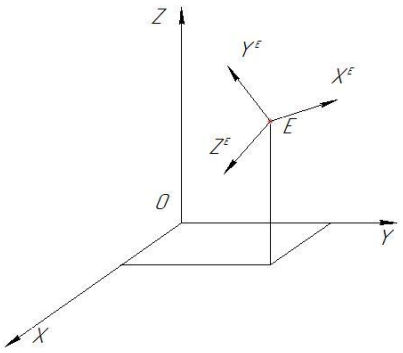



*Рисунок 8- Видовая система координат*

**Диффузионную модель отражения света**

Диффузное отражение – это вид отражения присущ матовым поверхностям . Матовой можно считать такую поверхность, размер шероховатостей которой уже настолько велик, что падающий луч рассеивается равномерно во все стороны.

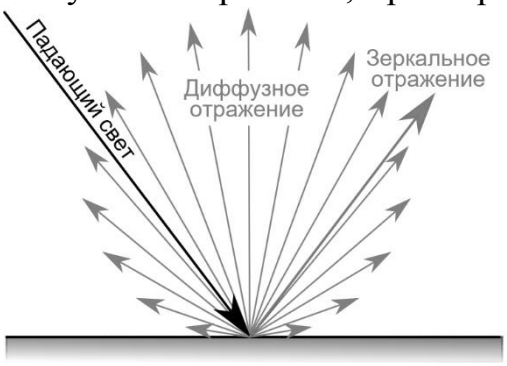



*Рисунок 9Диффузионное отражение света*

Интенсивность отражения света рассчитывается по формуле:

$$Id = I0Kd\cos\theta$$

I0 – интенсивность излучения источника.

Kd – коэффициент, который учитывает свойства материала поверхности (от 0 до 1).

θ – угол между направлением на точечный источник света и нормалью к поверхности.

Диффузное отражение определяется как косинус угла между вектором нормали к поверхности и некоторым направлением, определяемым вектором S (рис. 3).

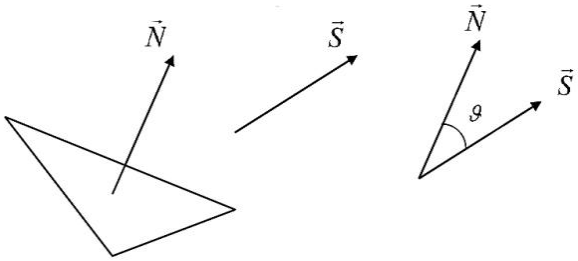



*Рисунок 10*

Источник света или наблюдатель находятся на бесконечности по отношению к некоторому элементу поверхности:

$$\vec{N} = \vec{N}(N_x, N_y, N_z)$$

вектор нормали к элементу поверхности :

$$\vec{S} = \vec{S}(S_x, S_y, S_z)$$

вектор определяющий некоторое направление в пространстве:

$$\cos\vartheta = \frac{\vec{S}\cdot\vec{N}}{\left|\vec{S}\right|\cdot\left|\vec{N}\right|} = \frac{S_xN_x + S_yN_y + S_zN_z}{\sqrt{S_x^2 + S_y^2 + S_z^2}\sqrt{N_x^2 + N_y^2 + N_z^2}}$$

## Методы закрашивания Гуро и Фонга

### Метод Гуро.

Предназначен для создания иллюзии гладкой криволинейной поверхности, описанной в виде многогранников или полигональной сетки с плоскими гранями.

Если каждая плоская грань имеет один постоянный цвет, определенный с учетом отражения, то различные цвета соседних граней очень заметны, и поверхность выглядит именно как многогранник.

Идея: закрашивания каждой плоской грани не одним цветом, а плавно изменяющимися оттенками, вычисляемыми путем интерполяции цветов примыкающих граней. Закрашивание граней по методу Гуро осуществляется в четыре этапа:

- вычисляются нормали к каждой грани
- определяются нормали в вершинах, нормаль в вершинах, нормаль в вершине определяется усреднением нормалей примыкающих граней рис. 4 и определяется по формуле:

$$\vec{N}_a = \frac{\vec{N}_1 + \vec{N}_2 + \vec{N}_3}{3}$$

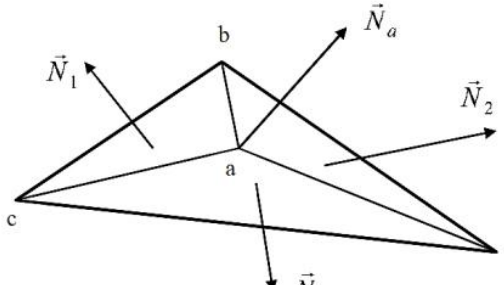



*Рисунок 11*

- на основе нормалей в вершинах вычисляются значения интенсивностей в вершинах согласно выбранной модели отражения света.
- закрашиваются полигоны граней цветом, соответствующим линейной интерполяции значений интенсивности в вершинах.

Заполнение контура грани горизонталями в экранных координатах

$$I_1 = I_b + (I_c - I_b)\frac{Y - Y_b}{Y_c - Y_b}$$

$$I_2 = I_b + (I_a - I_b)\frac{Y - Y_b}{Y_a - Y_b}$$

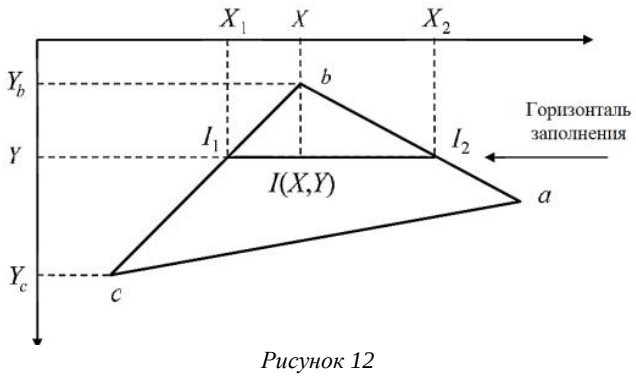



*Рисунок 12*

**Метод Фонга.**

Идея: используется интерполяция вектора нормали к поверхности вдоль видимого интервала на сканирующей строке внутри многоугольника, а не интерполяция интенсивности.

Интерполяция выполняется между начальной и конечной нормалями, которые сами тоже являются результатами интерполяции вдоль ребер многоугольника между нормалями в вершинах.

Нормали в вершинах, в свою очередь, вычисляются так же, как в методе закраски, построенном на основе интерполяции интенсивности.

Если скорость распространения света в двух средах отличается, то на границе этих сред происходит преломление падающего светового луча. Преломленный луч лежит в той же плоскости, что и векторы V и п, а угол падения связан с углом преломления

$$\eta_1 \sin\Theta_i = \eta_2 \sin\Theta_t$$

законом Снеллиуса.

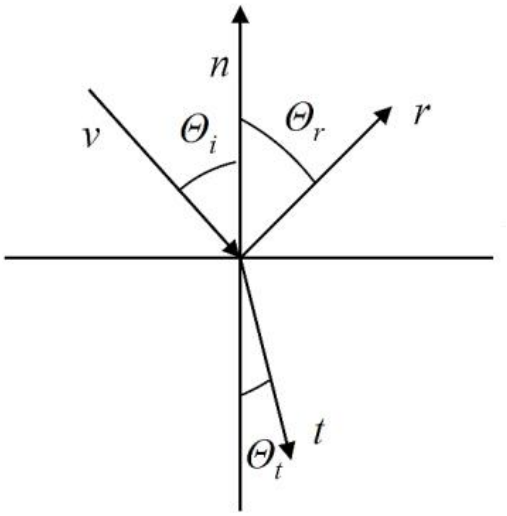



*Рисунок 13 Вектор преломления луча*

## Ход работы.

Суть лабораторной работы заключается в построении изображения пирамиды в двух видах: с удалением видимых граней и без удаления видимых граней.

Сделать нам это необходимо двумя разными инструментами: используя Mathcad и программируя на выбранном языке (в нашем случае это будет C++).

Исходные параметры такие:

- Координаты камеры в Сферической Системе Координат:
  - $r_v = 10$
  - $\theta_{vg}=315$ (градусы)
  - $\varphi_{vg}=45$ (градусы)
- Область отражения окна:
  - $X_{LW} = 100$; $Y_{LW} = 200$
  - $X_{HW} = 700$; $Y_{HW} = 500$
  - $X_{HW} = 800$; $Y_{HW} = 900$ – для Windows приложения
- Также следует задать координаты вершин нашей пирамиды, например:

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 3 | 0 | 0 | 1 |
| 3 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 3 | 3 | 3 |
| 1 | 1 | 1 | 1 | 1 | 1 |

## Часть I. Выполнение задания в Mathcad.

Прежде всего зададим нужные переменные: координаты вершин пирамиды и сферические координаты камеры.

$$PIR := \begin{pmatrix} 0 & 0 & 3 & 0 & 0 & 1 \\ 3 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 3 & 3 & 3 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \qquad PV := \begin{pmatrix} 10 \\ 315 \\ 45 \end{pmatrix}$$

$$r_v := PV_0 = 10 \qquad \phi_v := \frac{PV_1 \cdot \pi}{180} = 5.498 \qquad \theta_v := \frac{PV_2 \cdot \pi}{180} = 0.785$$

Также задаём координаты окна Windows:

$x_{LW} := 200$ $\quad y_{LW} := 100$ – левый верхний угол области отображения

$x_{HW} := 700$ $\quad y_{HW} := 500$ – правый нижний угол области отображения

Далее нам нужно перевести наши координаты из Видовых в Оконные. Для этого нам нужно построить матрицу пересчёта:

$$K_{view} := \begin{pmatrix} -\sin(\phi_v) & \cos(\phi_v) & 0 & 0 \\ -\cos(\theta_v)\cdot\cos(\phi_v) & -\cos(\theta_v)\cdot\sin(\phi_v) & \sin(\theta_v) & 0 \\ -\sin(\theta_v)\cdot\cos(\phi_v) & -\sin(\theta_v)\cdot\sin(\phi_v) & -\cos(\theta_v) & r_v \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Далее мы перемножаем нашу матрицу пересчёта и матрицу координат (именно в таком порядке!). А также формируем матрицу $P_{XY}$ из координат ***проекций*** вершин пирамиды на плоскость XY – плоскость в Видовой Системе Координат:

$$PIR\_V := K_{view}\cdot PIR$$

$$P_{XY} := \text{stack}\left[\left(PIR\_V^T\right)^{\langle 0\rangle^T}, \left(PIR\_V^T\right)^{\langle 1\rangle^T}, \left(PIR\_V^T\right)^{\langle 3\rangle^T}\right]$$

Первая строка этой матрицы является первой строкой получившейся матрицы PIR_V при перемножении матрицы пересчёта и матрицы координат вершин пирамиды. Вторая соответственно второй. А третья четвёртой.

Следующий шагом будет формирование *габаритной* области проекции пирамиды на плоскость XY видовой СК – картинную плоскость. Для этого найдем координаты левого верхнего угла и правого нижнего.

$$x_L := \min\left[\left(P_{XY}^T\right)^{\langle 0\rangle}\right] \qquad y_H := \max\left[\left(P_{XY}^T\right)^{\langle 1\rangle}\right]$$

**– левый верхний угол**

$$x_L = 0 \qquad y_H = 2.621$$

$$x_H := \max\left[\left(P_{XY}^T\right)^{\langle 0\rangle}\right] \qquad y_L := \min\left[\left(P_{XY}^T\right)^{\langle 1\rangle}\right]$$

**– правый нижний угол**

$$x_H = 2.121 \qquad y_L = -1.5$$

Далее нам нужно найти данные, необходимые для формирования матрицы пересчёта координат из плоскости XY видовой СК в оконную СК и сформировать матрицу пересчёта:

$$\Delta x_W := x_{HW} - x_{LW} \qquad \Delta x_W = 500$$

$$\Delta x := x_H - x_L \qquad \Delta x = 2.121$$

$$\Delta y_W := y_{HW} - y_{LW} \qquad \Delta y_W = 400$$

$$\Delta y := y_H - y_L \qquad \Delta y = 4.121$$

$$k_x := \frac{\Delta x_W}{\Delta x} \qquad k_x = 235.702$$

$$k_y := \frac{\Delta y_W}{\Delta y} \qquad k_y = 97.056$$

$$T_{SW} := \begin{pmatrix} k_x & 0 & x_{LW} - k_x \cdot x_L \\ 0 & -k_y & y_{HW} + k_y \cdot y_L \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 235.702 & 0 & 200 \\ 0 & -97.056 & 354.416 \\ 0 & 0 & 1 \end{pmatrix}$$ – матрица пересчета из ВСК в ОК

Вычисляем оконные координаты вершин пирамиды. Для этого умножим нашу сформированную матрицу пересчёта на матрицу координат XY Видовой СК. В полученную матрицу записываем числа, округлённые до целых:

$$M_W := T_{SW} \cdot P_{XY} \qquad \begin{pmatrix} x_a & x_b & x_c & x_d & x_e & x_f \\ y_a & y_b & y_c & y_d & y_e & y_f \\ q & q & q & q & q & q \end{pmatrix} := \overrightarrow{\text{round}(M_W)} = \left(\begin{array}{cccccc} 700 & 200 & 700 & 367 & 200 & 367 \\ 209 & 354 & 500 & 100 & 149 & 197 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{array}\right)$$

Чтобы построить пирамиду с удалением невидимых граней, нам требуется определить их видимость.

Для этого первым делом посчитаем декартовы координаты камеры – вектор, определяющий положение камеры:

$$R_C := \begin{pmatrix} r_v \cdot \sin(\theta_v) \cdot \cos(\phi_v) \\ r_v \cdot \sin(\theta_v) \cdot \sin(\phi_v) \\ r_v \cdot \cos(\theta_v) \end{pmatrix} = \begin{pmatrix} 5 \\ -5 \\ 7.071 \end{pmatrix}$$

Далее высчитываем декартовы координаты вектора внешней нормали к каждой грани пирамиды. Для этого следует найти координаты векторов, составляющие каждую из граней, например:

$$AC := \begin{pmatrix} C_0 \\ C_1 \\ C_2 \end{pmatrix} - \begin{pmatrix} A_0 \\ A_1 \\ A_2 \end{pmatrix}$$ - *координаты вектора AC (без единицы)*

$$AD := \begin{pmatrix} D_0 \\ D_1 \\ D_2 \end{pmatrix} - \begin{pmatrix} A_0 \\ A_1 \\ A_2 \end{pmatrix}$$ - *координаты вектора AD (без единицы)*

После этого, применив векторное произведение найти вектор внешней нормали:

$$N_{ADC} := AC \times AD$$

Порядок векторов определяется правосторонним движением. То есть для грани ABB'A':

$$A := \begin{bmatrix} 0 \\ 3 \\ 0 \end{bmatrix} \quad B := \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad C := \begin{bmatrix} 3 \\ 0 \\ 0 \end{bmatrix}$$

$$A' := \begin{bmatrix} 0 \\ 1 \\ 3 \end{bmatrix} \quad B' := \begin{bmatrix} 0 \\ 0 \\ 3 \end{bmatrix} \quad C' := \begin{bmatrix} 1 \\ 0 \\ 3 \end{bmatrix}$$

$$AB := B - A = \begin{bmatrix} 0 \\ -3 \\ 0 \end{bmatrix} \quad AA' := A' - A = \begin{bmatrix} 0 \\ -2 \\ 3 \end{bmatrix}$$

$$N_{ABB'A'} := AA' \times AB = \begin{bmatrix} 9 \\ 0 \\ 0 \end{bmatrix}$$

Видимость каждой из грани определяем нахождением косинуса угла между вектором внешней нормали и вектором камеры. Для этого нужно произведение векторов камеры и внешней нормали разделить на произведение их модулей (скалярное произведение):

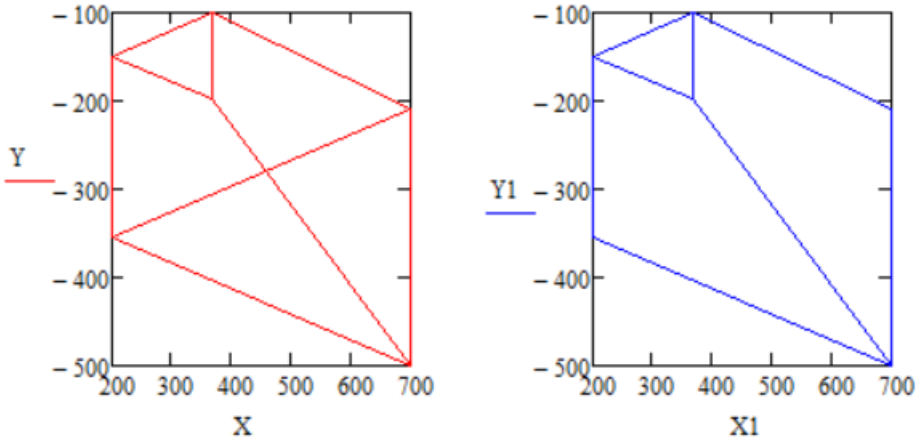
$$q1 := \frac{(R \cdot N_{ABB'A'})}{|R| \cdot |N_{ABB'A'}|} = 0.883$$

Так как у нас шестиугольная пирамида, то у нас имеется два основания. Определяем их видимость так: если верхнее основание видимо, то нижнее нет.

Далее формируем массивы координат для рисования наших пирамид. В случае с без удаления НГ – это все получившиеся координаты. В случае с удалением – те, что видимы.

$$X := (x_a\ x_b\ x_c\ x_a\ x_d\ x_e\ x_b\ x_e\ x_f\ x_c\ x_f\ x_d)^T \qquad X1 := (x_e\ x_b\ x_c\ x_f\ x_e\ x_d\ x_f\ x_c\ x_a\ x_d)^T$$

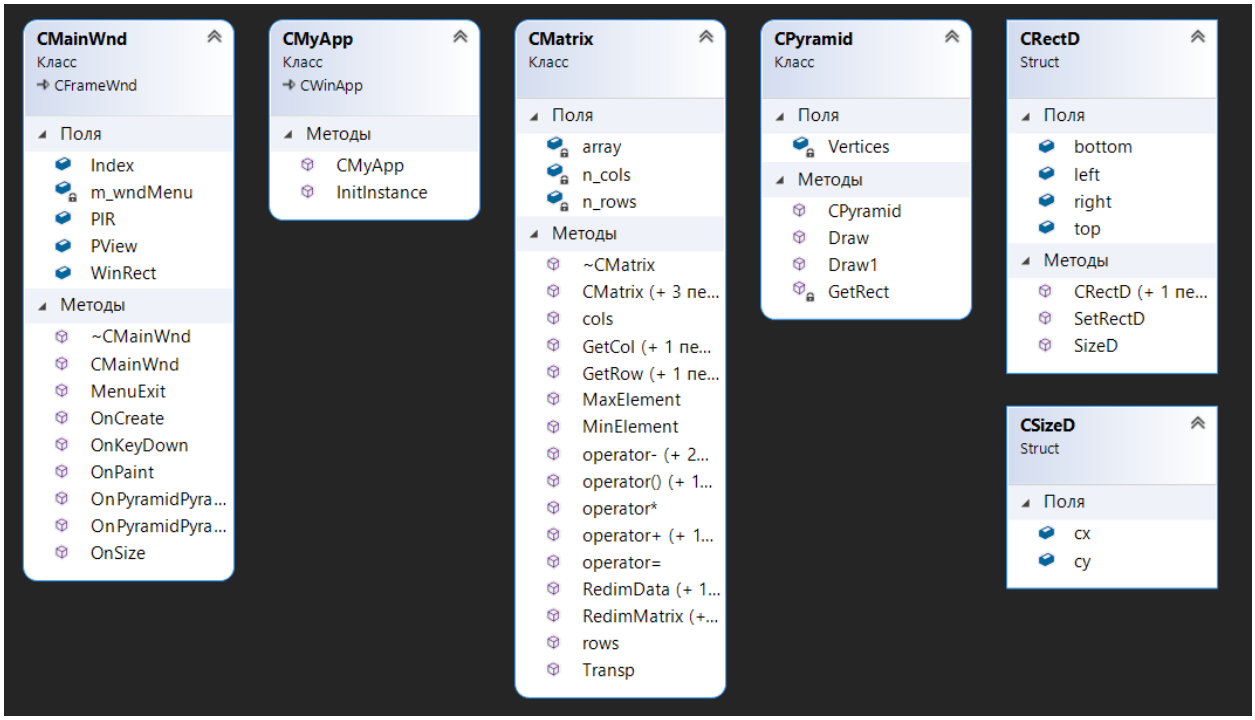
$$Y := -(y_a\ y_b\ y_c\ y_a\ y_d\ y_e\ y_b\ y_e\ y_f\ y_c\ y_f\ y_d)^T \qquad Y1 := -(y_e\ y_b\ y_c\ y_f\ y_e\ y_d\ y_f\ y_c\ y_a\ y_d)^T$$



## Часть II. Выполнение задания на C++

Теперь, когда мы понимаем принцип построения пирамид, следует реализовать их построение программно.

Наш проект будет иметь такую диаграмму классов:




Здесь мы имеем:

- **CMatrix** – класс, представляющий собой матрицу, в которой мы будем хранить наши координаты для построения пирамиды. Полями в нём являются: двумерный массив, а также значения количества строк и столбцов. Методы представляют собой нужные нам операции над матрицами.
- **CPyramid** – класс, представляющий собой пирамиду. В нём мы имеем поле Vertices (с типом данных CMatrix), где мы будем хранить вершины нашей пирамиды. Методами в нём являются функции, которые нужны по заданию – построение пирамиды с удалением НГ и без.
- **CRectD** – структура, представляющая собой прямоугольник. В ней мы будем хранить координаты окна нашего приложения.
- **CSizeD** – структура, представляющая собой одиночную координату.
- **CMainWind** и **CMyApp** – классы, реализующие окно нашего приложения (написано на MFC). В CMainWind мы имеем три важных поля – PIR (объект типа CPyramid) – наша пирамида, PView (объект

типа CMatrix) – координаты камеры в МСТ, и WinRect (объект CRectD) – координаты окна нашего приложения.

- Также здесь не указаны функции, которые нам нужны для выполнения задания: **CreateViewCoord**(), **VectorMult**() и **ScalarMult**() – они находятся в файле **LibGraph**, там же где и реализация структур **CRectD** и **CSizeD**.

Далее пройдемся по файловой структуре проекта: **LibPyramid** (реализация класса CPyramid), **LibGraph** (реализация CRectD и CSizeD), **Source** (реализация CMainWind и CMyApp), **stdafx** (Содержит стандартные подключения и константы), **CMatrix** (реализация класса CMatrix)

**LibGraph.h**

В этом файле мы имеем описание структур CSizeD, а также CRectD. Также мы здесь имеем функции:

- SpaceToWindow() – Возвращает матрицу пересчета координат из мировых в оконные.
- CreateViewCoord() – Создает матрицу пересчета точки из мировой системы координат в видовую.
- VectorMult() – векторное произведение векторов.
- ScalarMult() – скалярное произведение векторов.
- SphereToCart() – Преобразует сферические координаты PView точки в декартовы.

```
#ifndef LIBGRAPH
#define LIBGRAPH 1
const double pi=3.14159;

struct CSizeD
{
  double cx;
  double cy;
};

struct CRectD
 {
  double left;
  double top;
  double right;
  double bottom;
  CRectD(){left=top=right=bottom=0;};
  CRectD(double l,double t,double r,double b);
  void SetRectD(double l,double t,double r,double b);
  CSizeD SizeD();
 };

CMatrix SpaceToWindow(CRectD& rs,CRect& rw);
CMatrix CreateViewCoord(double r,double fi,double q);
CMatrix VectorMult(CMatrix& V1,CMatrix& V2);
double ScalarMult(CMatrix& V1,CMatrix& V2);
CMatrix SphereToCart(CMatrix& PView);
#endif
```

## LibGraph.cpp

```
#include "stdafx.h"
#include "CMatrix.h"
#include "LibGraph.h"
#include "math.h"
CRectD::CRectD(double l,double t,double r,double b)
{
 left=l;
 top=t;
 right=r;
 bottom=b;
}
void CRectD::SetRectD(double l,double t,double r,double b)
{
 left=l;
 top=t;
 right=r;
 bottom=b;
}
CSizeD CRectD::SizeD()
{
 CSizeD cz;
 cz.cx=fabs(right-left);       // Ширина прямоугольной области
 cz.cy=fabs(top-bottom);    // Высота прямоугольной области
 return cz;
}
CMatrix SpaceToWindow(CRectD& RS,CRect& RW)
// Функция обновлена
// Возвращает матрицу пересчета координат из мировых в оконные
// RS - область в мировых координатах - double
// RW - область в оконных координатах - int
{
 CMatrix M(3,3);
 CSize sz = RW.Size();       // Размер области в ОКНЕ
 int dwx=sz.cx;         // Ширина
 int dwy=sz.cy;         // Высота
 CSizeD szd=RS.SizeD(); // Размер области в МИРОВЫХ координатах

 double dsx=szd.cx;    // Ширина в мировых координатах
 double dsy=szd.cy;    // Высота в мировых координатах

 double kx=(double)dwx/dsx;   // Масштаб по x
 double ky=(double)dwy/dsy;   // Масштаб по y


        M(0,0)=kx;  M(0,1)=0;    M(0,2)=(double)RW.left-kx*RS.left;                    // Обновлено
 M(1,0)=0;   M(1,1)=-ky;  M(1,2)=(double)RW.bottom+ky*RS.bottom;          // Обновлено
 M(2,0)=0;   M(2,1)=0;    M(2,2)=1;
        return M;
 }
CMatrix VectorMult(CMatrix& V1,CMatrix& V2)
// Вычисляет векторное произведение векторов V1 и V2
//Векторное произведение — это псевдовектор, перпендикулярный плоскости, построенной по двум
сомножителям,
//являющийся результатом бинарной операции «векторное умножение» над векторами в трёхмерном
Евклидовом пространстве.
//Векторное произведение полезно для «измерения» перпендикулярности векторов — длина векторного
произведения двух векторов
//равна произведению их длин, если они перпендикулярны, и уменьшается до нуля, если векторы параллельны
либо антипараллельны.
{
        int b1=(V1.cols()==1)&&(V1.rows()==3);
        int b2=(V2.cols()==1)&&(V2.rows()==3);
        int b=b1&&b2;
```

```
	if(!b)
	{
		char* error="VectorMult: неправильные размерности векторов! ";
		MessageBox(NULL,error,"Ошибка",MB_ICONSTOP);
   exit(1);
	}
	CMatrix W(3);
	W(0)=V1(1)*V2(2)-V1(2)*V2(1);
	//double x=W(0);
	W(1)=-(V1(0)*V2(2)-V1(2)*V2(0));
	//double y=W(1);
	W(2)=V1(0)*V2(1)-V1(1)*V2(0);
	//double z=W(2);
	return W;
}
double ScalarMult(CMatrix& V1,CMatrix& V2)
// Вычисляет скалярное произведение векторов V1 и V2
//Скаля́рное произведе́ние — операция над двумя векторами, результатом которой является число (скаляр),
//не зависящее от системы координат и характеризующее длины векторов-сомножителей и угол между ними.
//Данной операции соответствует умножение длины вектора x на проекцию вектора y на вектор x. Эта операция
//обычно рассматривается как коммутативная и линейная по каждому сомножителю.
{
	int b1=(V1.cols()==1)&&(V1.rows()==3);
	int b2=(V2.cols()==1)&&(V2.rows()==3);
	int b=b1&&b2;
	if(!b)
	{
		char* error="ScalarMult: неправильные размерности векторов! ";
		MessageBox(NULL,error,"Ошибка",MB_ICONSTOP);
   exit(1);
	}
	double p=V1(0)*V2(0)+V1(1)*V2(1)+V1(2)*V2(2);
	return p;
}
CMatrix CreateViewCoord(double r,double fi,double q)
// Создает матрицу пересчета точки из мировой системы координат в видовую
// (r,fi,q)- координата ТОЧКИ НАБЛЮДЕНИЯ(начало видовой системы координат)
// в мировой сферической системе координат( углы fi и q в градусах)
{
	double fg=fmod(fi,360.0);
   double ff=(fg/180.0)*pi; // Перевод в радианы
	fg=fmod(q,360.0);
	double qq=(fg/180.0)*pi; // Перевод в радианы

	CMatrix VM(4,4);		// Матрица пересчета
	VM(0,0)=-sin(ff);   VM(0,1)=cos(ff);
	VM(1,0)=-cos(qq)*cos(ff); VM(1,1)=-cos(qq)*sin(ff);  VM(1,2)=sin(qq);
	VM(2,0)=-sin(qq)*cos(ff);  VM(2,1)=-sin(qq)*sin(ff);   VM(2,2)=-cos(qq); VM(2,3)=r;
	VM(3,3)=1;
	return VM;
}

CMatrix SphereToCart(CMatrix& PView)
// Преобразует сферические координаты PView  точки в декартовы
// PView(0) - r
// PView(1) - fi - азимут(отсчет от оси X), град.
// PView(2) - q - угол(отсчетот оси Z), град.
// Результат: R(0)- x, R(1)- y, R(2)- z
{
	CMatrix R(3);
	double r=PView(0);
	double fi=PView(1);                                                    // Градусы
	double q=PView(2);                                                     // Градусы
	double fi_rad=(fi/180.0)*pi;		// Перевод fi в радианы
	double q_rad=(q/180.0)*pi;		// Перевод q в радианы
	R(0)=r*sin(q_rad)*cos(fi_rad);		// x- координата точки наблюдения
```

```
        R(1)=r*sin(q_rad)*sin(fi_rad);        // y- координата точки наблюдения
        R(2)=r*cos(q_rad);                                    // z- координата точки наблюдения
        return R;
}
```

## Lab05.cpp

```
#include "afxwin.h"                                                    // MFC Основные и стандартные
компоненты
#include "afxext.h"                                                    // MFC Расширения
#include "resource.h"                                          // Идентификаторы ресурсов
#include "CMatrix.h"
#include "LibGraph.h"
#include "LibPyramid.h"
#include "math.h"

#define IDR_MENU1                101

class CMainWnd : public CFrameWnd
{
public:
        CMainWnd();
        int OnCreate(LPCREATESTRUCT lpCreateStruct);
        void OnPaint();
        void MenuExit();
        ~CMainWnd();

        CPyramid PIR;    // Координаты вершин пирамиды
        CRect WinRect;   // Область отображения
        CMatrix PView;   // Координаты положения камеры
        int Index;
        afx_msg void OnPyramidPyramid1();          // без удаления невидимых линий
        afx_msg void OnPyramidPyramid2();          // с удалением невидимых линий
        afx_msg void OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags);
        afx_msg void OnSize(UINT nType, int cx, int cy);

private:
        CMenu m_wndMenu;
        DECLARE_MESSAGE_MAP();
};

BEGIN_MESSAGE_MAP(CMainWnd, CFrameWnd)
        ON_WM_PAINT()
        ON_WM_CREATE()

        ON_COMMAND(ID_PYRAMID_PIRAMID1,&CMainWnd::OnPyramidPyramid1)
        ON_WM_KEYDOWN()
        ON_WM_SIZE()
        ON_COMMAND(ID_PYRAMID_PIRAMID2,&CMainWnd::OnPyramidPyramid2)

        ON_COMMAND(ID_FILE_EXIT,MenuExit)
END_MESSAGE_MAP()

void CMainWnd::OnPaint()
{
        CPaintDC dc(this);   // Получить контекст устройства

        if(Index==1)PIR.Draw(dc,PView,WinRect);
        if(Index==2)PIR.Draw1(dc,PView,WinRect);
}

void CMainWnd::OnPyramidPyramid1()
{
        PView(0)= 10;    PView(1)=315;    PView(2)=45;
        Index=1;
        Invalidate();
```

```
}

void CMainWnd::OnPyramidPyramid2()
{
	PView(0)= 10;	PView(1)=315;	PView(2)=45;
	Index=2;
	Invalidate();
}

void CMainWnd::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)
{
	if((Index==1)||(Index==2))
 {
	switch(nChar)
	{
	 case VK_UP:
		{
		 double d=PView(2)-5;		// Изменение угла тета
		 if(d>=0)PView(2)=d;
		 break;
		}
	 case VK_DOWN:
		{
		 double d=PView(2)+5;		// Изменение угла тета
		 if(d<=180)PView(2)=d;
		 break;
		}
	 case VK_LEFT:
		{
		 double d=PView(1)-10;		// Изменение угла фи
		 if(d>=-180)PView(1)=d;
		 else PView(1)=d+360;
		 break;
		}
	 case VK_RIGHT:
		{
		 double d=PView(1)+10;		// Изменение угла фи
		 if(d<=180)PView(1)=d;
		 else PView(1)=d-360;
		 break;
		}
	}
	Invalidate();
 }
}

void CMainWnd::OnSize(UINT nType, int cx, int cy)
{
	WinRect.SetRect(100,100,cx - 100, cy - 50);
}

void CMainWnd::MenuExit()
{
	DestroyWindow();
}

int CMainWnd::OnCreate(LPCREATESTRUCT lpCreateStruct)
 {
	 if (CFrameWnd::OnCreate(lpCreateStruct) == -1)  return -1;
	 m_wndMenu.LoadMenu(IDR_MENU1);
	 SetMenu(&m_wndMenu);
	 Index=0;
	 PView.RedimMatrix(3);
	 return 0;
}
```

```
CMainWnd::CMainWnd()
{
	Create(NULL,"Lab05",(WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU | WS_MINIMIZEBOX |
WS_MAXIMIZEBOX) &~(WS_THICKFRAME),rectDefault,NULL,NULL);
}

CMainWnd::~CMainWnd()
{
}

class CMyApp : public CWinApp
{
public:
	CMyApp();
	virtual BOOL InitInstance();
};

CMyApp::CMyApp()
{}

BOOL CMyApp::InitInstance()
{
	m_pMainWnd=new CMainWnd();
	ASSERT(m_pMainWnd);
	m_pMainWnd->ShowWindow(SW_SHOW);
	m_pMainWnd->UpdateWindow();
	return TRUE;
};

CMyApp theApp;
```

## LibPyramid.h

```
#include "CMatrix.h"
#include "LibGraph.h"
class CPyramid
{
		CMatrix Vertices;
		void GetRect(CMatrix& Vert,CRectD& RectView);
	public:
		CPyramid();
		void Draw(CDC& dc,CMatrix& P,CRect& RW);
		void Draw1(CDC& dc,CMatrix& P,CRect& RW);
};
```

## LibPyramid.cpp

```
#include "stdafx.h"
#include "LibPyramid.h"
#include "CMatrix.h"
#include "LibGraph.h"
CPyramid::CPyramid()
{
	Vertices.RedimMatrix(4,8);
	// Координаты вершин - столбцы
	Vertices(0,0)=6;		// A: x=6,y=0,z=0
	Vertices(0,3)=2;		// D: x=2,y=0,z=6
	Vertices(2,3)=6;

	Vertices(1,1)=-6;		// B: x=0,y=-6,z=0
	Vertices(1,4)=-2;		// E: x=0,y=-2,z=6
	Vertices(2,4)=6;

	Vertices(0,2)=-6;		// C: x=-6,y=0,z=0
```

```
	Vertices(0,5)=-2;		// F: x=-2,y=0,z=6
	Vertices(2,5)=6;
}

// Рисует пирамиду БЕЗ удаления невидимых ребер
void CPyramid::Draw(CDC& dc,CMatrix& PView,CRect& RW)
{
	CMatrix ViewCart=SphereToCart(PView);					// В декартовы координаты точки наблюдения
	CMatrix MV=CreateViewCoord(PView(0),PView(1),PView(2));		// Матрица пересчета из МСК в видовую СК
	CMatrix ViewVert=MV*Vertices;				// Координаты вершин пирамиды в видовой СК
	CRectD RectView;
	GetRect(ViewVert,RectView);					// Получаем охватывающий прямоугольник
	CMatrix MW=SpaceToWindow(RectView,RW);		// Матрица пересчета в ОСК
	// Готовим массив оконных координат для рисования
	CPoint MasVert[6];							// Массив оконных координат вершин
	CMatrix V(3);
	V(2)=1;
	// Цикл по количеству вершин - вычисляем оконные координаты вершин
	for(int i=0;i<6;i++)
	{
		V(0)=ViewVert(0,i);			// x
		V(1)=ViewVert(1,i);			// y
		V=MW*V;						// Оконные координаты точки
		MasVert[i].x=(int)V(0);
		MasVert[i].y=(int)V(1);
	}
	// Рисуем
	CPen Pen(PS_SOLID, 2, RGB(0, 0, 255));
	CPen* pOldPen =dc.SelectObject(&Pen);
	dc.MoveTo(MasVert[2]);
	for(int i = 0; i < 3; i++) //нижнее основание
	{
		dc.LineTo(MasVert[i]);
	}
	dc.MoveTo(MasVert[5]);
	for(int i = 3; i < 6; i++) //верхнее основание
	{
		dc.LineTo(MasVert[i]);
	}
	for(int i=0;i<3;i++)		//ребра
	{
		dc.MoveTo(MasVert[i]);
		dc.LineTo(MasVert[i+3]);
	}

	// Координаты центра O	пересечения диагоналей основания
	int A1x = (MasVert[1].x + MasVert[2].x) / 2;
	int A1y = (MasVert[1].y + MasVert[2].y) / 2;
	int B1x = (MasVert[0].x + MasVert[2].x) / 2;
	int B1y = (MasVert[0].y + MasVert[2].y) / 2;
	int C1x = (MasVert[1].x + MasVert[0].x) / 2;
	int C1y = (MasVert[1].y + MasVert[0].y) / 2;
	CPen Pen1(PS_DASH, 1, RGB(120, 60, 0));
	dc.SelectObject(&Pen1);
	dc.MoveTo(MasVert[0]);	// Перо на вершину A
	dc.LineTo(A1x, A1y);	// Диагональ
	dc.MoveTo(MasVert[1]);	// Перо на вершину B
	dc.LineTo(B1x, B1y);	// Диагональ
	dc.MoveTo(MasVert[2]);	// Перо на вершину C
	dc.LineTo(C1x, C1y);
	A1x = (MasVert[4].x + MasVert[5].x) / 2;
	A1y = (MasVert[4].y + MasVert[5].y) / 2;
```

```
	B1x = (MasVert[3].x + MasVert[5].x) / 2;
	B1y = (MasVert[3].y + MasVert[5].y) / 2;
	C1x = (MasVert[4].x + MasVert[3].x) / 2;
	C1y = (MasVert[4].y + MasVert[3].y) / 2;
	dc.MoveTo(MasVert[3]);   // Перо на вершину A
	dc.LineTo(A1x, A1y);     // Диагональ
	dc.MoveTo(MasVert[4]);   // Перо на вершину B
	dc.LineTo(B1x, B1y);     // Диагональ
	dc.MoveTo(MasVert[5]);   // Перо на вершину C
	dc.LineTo(C1x, C1y);
	dc.SelectObject(pOldPen);
}

void CPyramid::Draw1(CDC& dc,CMatrix& PView,CRect& RW)
{
	CMatrix ViewCart=SphereToCart(PView);
	CMatrix MV=CreateViewCoord(PView(0),PView(1),PView(2));
	CMatrix ViewVert=MV*Vertices;
	CRectD RectView;
	GetRect(ViewVert,RectView);
	CMatrix MW=SpaceToWindow(RectView,RW);

	CPoint MasVert[6];
	CMatrix V(3);
	V(2)=1;
	for(int i=0;i<6;i++)
	{
		V(0)=ViewVert(0,i); // x
		V(1)=ViewVert(1,i); // y

		V=MW*V;
		MasVert[i].x=(int)V(0);
		MasVert[i].y=(int)V(1);
	}

	CPen Pen(PS_SOLID, 2, RGB(0, 0, 255));
	CPen* pOldPen = dc.SelectObject(&Pen);

	CBrush Brus(RGB(120, 60, 0));                    // Для нижнего основания
	CBrush* pOldBrush =dc.SelectObject(&Brus);

	CMatrix R1(3),R2(3),VN(3);
	double sm;
	for(int i=0;i<3;i++)                                  // рисуем ребра без невидимых
	{
		CMatrix VE=Vertices.GetCol(i + 3,0,2);
		int k;
		if(i==2) k=0;
		else k=i+1;
		R1=Vertices.GetCol(i,0,2);
		R2=Vertices.GetCol(k,0,2);
		CMatrix V1=R2-R1;
		CMatrix V2=VE-R1;
		VN=VectorMult(V2,V1);
		sm=ScalarMult(VN,ViewCart);
		if (sm >= 0)
		{
			dc.MoveTo(MasVert[i]);
			dc.LineTo(MasVert[k]);
			dc.LineTo(MasVert[k + 3]);
			dc.LineTo(MasVert[i + 3]);
			dc.LineTo(MasVert[i]);
		}
	}

	if(ViewCart(2)<0)
```

```
                dc.Polygon(MasVert, 3);     // нижнее основание
        else
        {
                CBrush *topBrush = new CBrush((COLORREF)0x1fffff);
                dc.SelectObject(topBrush);
                dc.Polygon(MasVert + 3, 3);             // верхнее основание
        }

        dc.SelectObject(pOldPen);
        dc.SelectObject(pOldBrush);
}

// Вычисляет координаты прямоугольника,охватывающего проекцию
// пирамиды на плоскость XY в ВИДОВОЙ системе координат
void CPyramid::GetRect(CMatrix& Vert,CRectD& RectView)
{
        CMatrix V=Vert.GetRow(0);                       // x - координаты
        double xMin=V.MinElement();
        double xMax=V.MaxElement();
        V=Vert.GetRow(1);                                       // y - координаты
        double yMin=V.MinElement();
        double yMax=V.MaxElement();
        RectView.SetRectD(xMin,yMax,xMax,yMin);
}
```

## stdafx.h

```
// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently,
// but are changed infrequently
#pragma once

#ifndef _SECURE_ATL
#define _SECURE_ATL 1
#endif

#ifndef VC_EXTRALEAN
#define VC_EXTRALEAN            // Exclude rarely-used stuff from Windows headers
#endif

#define _ATL_CSTRING_EXPLICIT_CONSTRUCTORS      // some CString constructors will be explicit

// turns off MFC's hiding of some common and often safely ignored warning messages
#define _AFX_ALL_WARNINGS

#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions

#ifndef _AFX_NO_OLE_SUPPORT
#include <afxdtctl.h>           // MFC support for Internet Explorer 4 Common Controls
#endif
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>             // MFC support for Windows Common Controls
#endif // _AFX_NO_AFXCMN_SUPPORT
```

## CMatrix.h

```
#ifndef CMATRIXH
# define CMATRIXH 1
class CMatrix
{
        double **array;
        int n_rows;                                                     // Число строк
        int n_cols;                                                     // Число столбцов
```

```
public:
        CMatrix();                                                          // Конструктор по умолчанию (1
на 1)
        CMatrix(int,int);                               // Конструктор
        CMatrix(int);                                                       // Конструктор -вектора (один столбец)
        CMatrix(const CMatrix&);                        // Конструктор копирования
        ~CMatrix();
        double &operator()(int,int);       // Выбор элемента матрицы по индексу
        double &operator()(int);           // Выбор элемента вектора по индексу
        CMatrix operator-();                                // Оператор "-"
        CMatrix operator=(const CMatrix&);              // Оператор "Присвоить":    M1=M2
        CMatrix operator*(CMatrix&);       // Оператор "Произведение": M1*M2
        CMatrix operator+(CMatrix&);            // Оператор "+": M1+M2
        CMatrix operator-(CMatrix&);            // Оператор "-": M1-M2
        CMatrix operator+(double);                      // Оператор "+": M+a
        CMatrix operator-(double);              // Оператор "-": M-a
        int rows()const{return n_rows;} ;  // Возвращает число строк
        int cols()const{return n_cols;};   // Возвращает число строк
        CMatrix Transp();                               // Возвращает матрицу,транспонированную к текущей
        CMatrix GetRow(int);                            // Возвращает строку по номеру
        CMatrix GetRow(int,int,int);
        CMatrix GetCol(int);                            // Возвращает столбец по номеру
        CMatrix GetCol(int,int,int);
        CMatrix RedimMatrix(int,int);           // Изменяет размер матрицы с уничтожением данных
        CMatrix RedimData(int,int);        // Изменяет размер матрицы с сохранением данных,
                                //которые можно сохранить
        CMatrix RedimMatrix(int);       // Изменяет размер матрицы с уничтожением данных
        CMatrix RedimData(int);          // Изменяет размер матрицы с сохранением данных,
                        //которые можно сохранить
        double MaxElement();                                    // Максимальный элемент матрицы
        double MinElement();                                    // Минимальный элемент матрицы
};
#endif
```

## CMatrix.cpp

```
#include "stdafx.h"
#include "CMatrix.h"

CMatrix::CMatrix()
{
 n_rows=1;
 n_cols=1;
 array=new double*[n_rows];
 for(int i=0;i<n_rows;i++) array[i]=new double[n_cols];
 for(int i=0;i<n_rows;i++)
   for(int j=0;j<n_cols;j++) array[i][j]=0;
 }

//--------------------------------------------------------------------------
CMatrix::CMatrix(int Nrow,int Ncol)
// Nrow - число строк
// Ncol - число столбцов
{
 n_rows=Nrow;
 n_cols=Ncol;
 array=new double*[n_rows];
 for(int i=0;i<n_rows;i++) array[i]=new double[n_cols];
 for(int i=0;i<n_rows;i++)
   for(int j=0;j<n_cols;j++) array[i][j]=0;
 }

//--------------------------------------------------------------------------
CMatrix::CMatrix(int Nrow)  //Вектор
```

```
// Nrow - число строк
{
 n_rows=Nrow;
 n_cols=1;
 array=new double*[n_rows];
 for(int i=0;i<n_rows;i++) array[i]=new double[n_cols];
 for(int i=0;i<n_rows;i++)
   for(int j=0;j<n_cols;j++) array[i][j]=0;
 }
//---------------------------------------------------------------------------
CMatrix::~CMatrix()
{
 for(int i=0;i<n_rows;i++) delete array[i];
 delete array;
}

//---------------------------------------------------------------------------
double &CMatrix::operator()(int i,int j)
// i - номер строки
// j - номер столбца
{
if ((i>n_rows-1)||(j>n_cols-1))     //  проверка выхода за диапазон
   {
                        TCHAR* error=_T("CMatrix::operator(int,int): выход индекса за границу диапазона ");
                        MessageBox(NULL,error,_T("Ошибка"),MB_ICONSTOP);
    exit(1);
   }
return array[i][j];
}

//---------------------------------------------------------------------------
double &CMatrix::operator()(int i)
// i - номер строки для вектора
{
if (n_cols>1)     //  Число столбцов больше одного
   {
    char* error="CMatrix::operator(int): объект не вектор - число столбцов больше 1 ";
                        MessageBox(NULL,error,"Ошибка",MB_ICONSTOP);
    exit(1);
   }
if (i>n_rows-1)     //  проверка выхода за диапазон
   {
    TCHAR* error=TEXT("CMatrix::operator(int): выход индекса за границу диапазона ");
                        MessageBox(NULL,error,TEXT("Ошибка"),MB_ICONSTOP);
    exit(1);
   }
return array[i][0];
}
//---------------------------------------------------------------------------
CMatrix CMatrix::operator-()
// Оператор -M
{
            CMatrix Temp(n_rows,n_cols);
  for(int i=0;i<n_rows;i++)
            for(int j=0;j<n_cols;j++) Temp(i,j)=-array[i][j];
  return Temp;
}

//---------------------------------------------------------------------------
CMatrix CMatrix::operator+(CMatrix& M)
// Оператор M1+M2
{
            int bb=(n_rows==M.rows())&&(n_cols==M.cols());
            if(!bb)
            {
                        char* error="CMatrix::operator(+): несоответствие размерностей матриц ";
```

```
                    MessageBox(NULL,error,"Ошибка",MB_ICONSTOP);
   exit(1);
            }
            CMatrix Temp(*this);
 for(int i=0;i<n_rows;i++)
            for(int j=0;j<n_cols;j++) Temp(i,j)+=M(i,j);
 return Temp;
}
//---------------------------------------------------------------------------
CMatrix CMatrix::operator-(CMatrix& M)
// Оператор M1-M2
{
            int bb=(n_rows==M.rows())&&(n_cols==M.cols());
            if(!bb)
            {
                    char* error="CMatrix::operator(-): несоответствие размерностей матриц ";
                    MessageBox(NULL,error,"Ошибка",MB_ICONSTOP);
   exit(1);
            }
            CMatrix Temp(*this);
            for(int i=0;i<n_rows;i++)
            for(int j=0;j<n_cols;j++) Temp(i,j)-=M(i,j);
 return Temp;
}
//---------------------------------------------------------------------------
CMatrix CMatrix::operator*(CMatrix& M)
// Умножение на матрицу M
{
double sum;
int nn=M.rows();
int mm=M.cols();
CMatrix Temp(n_rows,mm);
if (n_cols==nn)
  {
   for (int i=0;i<n_rows;i++)
       for (int j=0;j<mm;j++)
         {
         sum=0;
         for (int k=0;k<n_cols;k++) sum+=(*this)(i,k)*M(k,j);
         Temp(i,j)=sum;
         }
  }
else
 {
   TCHAR* error=TEXT("CMatrix::operator*: несоответствие размерностей матриц ");
   MessageBox(NULL,error,TEXT("Ошибка"),MB_ICONSTOP);
   exit(1);
  }
return Temp;
}

//---------------------------------------------------------------------------
CMatrix CMatrix::operator=(const CMatrix& M)
// Оператор присваивания M1=M
{
if (this==&M) return *this;
int nn=M.rows();
int mm=M.cols();
if ((n_rows==nn)&&(n_cols==mm))
 {
for (int i=0;i<n_rows;i++)
    for (int j=0;j<n_cols;j++) array[i][j]=M.array[i][j];
 }
else   // для ошибки размерностей
  {
   TCHAR* error=TEXT("CMatrix::operator=: несоответствие размерностей матриц");
```

```
   MessageBox(NULL,error,TEXT("Ошибка"),MB_ICONSTOP);
   exit(1);
  }
return *this;
}

//---------------------------------------------------------------------------
CMatrix::CMatrix(const CMatrix &M) // Конструктор копирования
{
            n_rows=M.n_rows;
            n_cols=M.n_cols;
            array=new double*[n_rows];
            for(int i=0;i<n_rows;i++) array[i]=new double[n_cols];
            for(int i=0;i<n_rows;i++)
    for(int j=0;j<n_cols;j++) array[i][j]=M.array[i][j];
}

//---------------------------------------------------------------------------
CMatrix CMatrix::operator+(double x)
// Оператор M+x, где M - матрица, x - число
{
  CMatrix Temp(*this);
  for(int i=0;i<n_rows;i++)
            for(int j=0;j<n_cols;j++) Temp(i,j)+=x;
  return Temp;
}

//---------------------------------------------------------------------------
CMatrix CMatrix::operator-(double x)
// Оператор M+x, где M - матрица, x - число
{
  CMatrix Temp(*this);
  for(int i=0;i<n_rows;i++)
            for(int j=0;j<n_cols;j++) Temp(i,j)-=x;
  return Temp;
}
//---------------------------------------------------------------------------
CMatrix CMatrix::Transp()
// Возвращает матрицу,транспонированную к (*this)
{
  CMatrix Temp(n_cols,n_rows);
  for (int i=0;i<n_cols;i++)
     for (int j=0;j<n_rows;j++) Temp(i,j)=array[j][i];
  return Temp;
}

//---------------------------------------------------------------------------
CMatrix CMatrix::GetRow(int k)
// Возвращает строку матрицы по номеру k
{
            if(k>n_rows-1)
            {
                        char* error="CMatrix::GetRow(int k): параметр k превышает число строк ";
                        MessageBox(NULL,error,"Ошибка",MB_ICONSTOP);
    exit(1);
            }
            CMatrix M(1,n_cols);
            for(int i=0;i<n_cols;i++)M(0,i)=(*this)(k,i);
            return M;
}
//---------------------------------------------------------------------------
CMatrix CMatrix::GetRow(int k,int n,int m)
// Возвращает подстроку из строки матрицы с номером k
// n - номер первого элемента в строке
// m - номер последнего элемента в строке
{
```

```
	int b1=(k>=0)&&(k<n_rows);
	int b2=(n>=0)&&(n<=m);
	int b3=(m>=0)&&(m<n_cols);
	int b4=b1&&b2&&b3;
	if(!b4)
	{
		char* error="CMatrix::GetRow(int k,int n, int m):ошибка в параметрах ";
		MessageBox(NULL,error,"Ошибка",MB_ICONSTOP);
  exit(1);
	}
	int nCols=m-n+1;
	CMatrix M(1,nCols);
	for(int i=n;i<=m;i++)M(0,i-n)=(*this)(k,i);
	return M;
}

//---------------------------------------------------------------------------
CMatrix CMatrix::GetCol(int k)
// Возвращает столбец матрицы по номеру k
{
	if(k>n_cols-1)
	{
		char* error="CMatrix::GetCol(int k): параметр k превышает число столбцов ";
		MessageBox(NULL,error,"Ошибка",MB_ICONSTOP);
  exit(1);
	}
	CMatrix M(n_rows,1);
	for(int i=0;i<n_rows;i++)M(i,0)=(*this)(i,k);
	return M;
}
//---------------------------------------------------------------------------
CMatrix CMatrix::GetCol(int k,int n, int m)
// Возвращает подстолбец из столбца матрицы с номером k
// n - номер первого элемента в столбце
// m - номер последнего элемента в столбце
{
	int b1=(k>=0)&&(k<n_cols);
	int b2=(n>=0)&&(n<=m);
	int b3=(m>=0)&&(m<n_rows);
	int b4=b1&&b2&&b3;
	if(!b4)
	{
		char* error="CMatrix::GetCol(int k,int n, int m):ошибка в параметрах ";
		MessageBox(NULL,error,"Ошибка",MB_ICONSTOP);
  exit(1);
	}
	int nRows=m-n+1;
	CMatrix M(nRows,1);
	for(int i=n;i<=m;i++)M(i-n,0)=(*this)(i,k);
	return M;
}

//---------------------------------------------------------------------------
CMatrix CMatrix::RedimMatrix(int NewRow,int NewCol)
// Изменяет размер матрицы с уничтожением данных
// NewRow - новое число строк
// NewCol - новое число столбцов
{
	for(int i=0;i<n_rows;i++) delete array[i];
	delete array;
	n_rows=NewRow;
	n_cols=NewCol;
	array=new double*[n_rows];
	for(int i=0;i<n_rows;i++) array[i]=new double[n_cols];
	for(int i=0;i<n_rows;i++)
		for(int j=0;j<n_cols;j++) array[i][j]=0;
```

```
	return (*this);
}

//---------------------------------------------------------------------------
CMatrix CMatrix::RedimData(int NewRow,int NewCol)
// Изменяет размер матрицы с сохранением данных, которые можно сохранить
// NewRow - новое число строк
// NewCol - новое число столбцов
{
	CMatrix Temp=(*this);
	this->RedimMatrix(NewRow,NewCol);
	int min_rows=Temp.rows()<(*this).rows()?Temp.rows():(*this).rows();
	int min_cols=Temp.cols()<(*this).cols()?Temp.cols():(*this).cols();
	for(int i=0;i<min_rows;i++)
		for(int j=0;j<min_cols;j++) (*this)(i,j)=Temp(i,j);
	return (*this);
}

//---------------------------------------------------------------------------
CMatrix CMatrix::RedimMatrix(int NewRow)
// Изменяет размер матрицы с уничтожением данных
// NewRow - новое число строк
// NewCol=1
{
	for(int i=0;i<n_rows;i++) delete array[i];
	delete array;
	n_rows=NewRow;
	n_cols=1;
	array=new double*[n_rows];
	for(int i=0;i<n_rows;i++) array[i]=new double[n_cols];
	for(int i=0;i<n_rows;i++)
		for(int j=0;j<n_cols;j++) array[i][j]=0;
	return (*this);
}

//---------------------------------------------------------------------------
CMatrix CMatrix::RedimData(int NewRow)
// Изменяет размер матрицы с сохранением данных, которые можно сохранить
// NewRow - новое число строк
// NewCol=1
{
	CMatrix Temp=(*this);
	this->RedimMatrix(NewRow);
	int min_rows=Temp.rows()<(*this).rows()?Temp.rows():(*this).rows();
	for(int i=0;i<min_rows;i++)(*this)(i)=Temp(i);
	return (*this);
}
//---------------------------------------------------------------------------
double CMatrix::MaxElement()
// Максимальное значение элементов матрицы
{
	double max=(*this)(0,0);
	for(int i=0;i<(this->rows());i++)
	  for(int j=0;j<(this->cols());j++) if ((*this)(i,j)>max) max=(*this)(i,j);
	return max;
}

//---------------------------------------------------------------------------
double CMatrix::MinElement()
// Минимальное значение элементов матрицы
{
	double min=(*this)(0,0);
	for(int i=0;i<(this->rows());i++)
	  for(int j=0;j<(this->cols());j++) if ((*this)(i,j)<min) min=(*this)(i,j);
	return min;
}
```